Fwrap:
Fortran for speed
Python for comfort

Kurt W. Smith
Dag Sverre Seljebotn

University of Wisconsin-Madison
Center for Magnetic Self-Organization

SciPy Conference, Caltech

21 August 2009

Why would you be interested?

Why would you be interested?

The same reasons that you use `f2py`:

Why would you be interested?

The same reasons that you use `f2py`:

- Automatically wraps Fortran code in Python.

Why would you be interested?

The same reasons that you use `f2py`:

- Automatically wraps Fortran code in Python.
- Takes care of creating the extension module for you.

Why would you be interested?

The same reasons that you use `f2py`:

- Automatically wraps Fortran code in Python.
- Takes care of creating the extension module for you.
- Give indispensable Fortran 77 code a Python interface.

Why would you be interested?

The same reasons that you use `f2py`:

- Automatically wraps Fortran code in Python.
- Takes care of creating the extension module for you.
- Give indispensable Fortran 77 code a Python interface.
- You want to make use of the dynamism of Python to ease testing of a large Fortran 77 or 9x codebase

Why would you be interested?

The same reasons that you use `f2py`:

- Automatically wraps Fortran code in Python.
- Takes care of creating the extension module for you.
- Give indispensable Fortran 77 code a Python interface.
- You want to make use of the dynamism of Python to ease testing of a large Fortran 77 or 9x codebase
- Hybrid numerical programming – Fortran for speed, Python for comfort.

`fwrap` = `f2py` **PLUS...**

`fwrap` = `f2py` **PLUS...**

Updated features (NB: not all implemented yet!)

`fwrap` = `f2py` **PLUS...**

Updated features (NB: not all implemented yet!)

- Greater portability – wrap once & distribute, compile on client.

`fwrap` = `f2py` **PLUS...**

Updated features (NB: not all implemented yet!)

- Greater portability – wrap once & distribute, compile on client.
- Extension module is Python 2.x & Python 3.x compatible.

`fwrap` = `f2py` **PLUS...**

Updated features (NB: not all implemented yet!)

- Greater portability – wrap once & distribute, compile on client.
- Extension module is Python 2.x & Python 3.x compatible.
- Automatic resolution of kind-type-parameters:

`fwrap` = `f2py` **PLUS...**

Updated features (NB: not all implemented yet!)

- Greater portability – wrap once & distribute, compile on client.
- Extension module is Python 2.x & Python 3.x compatible.
- Automatic resolution of kind-type-parameters:

```
integer(selected_int_kind(0))
complex(kind=DCPX)
real*8
```

`fwrap` = `f2py` **PLUS...**

Updated features (NB: not all implemented yet!)

- Greater portability – wrap once & distribute, compile on client.
- Extension module is Python 2.x & Python 3.x compatible.
- Automatic resolution of kind-type-parameters:

```
integer(selected_int_kind(0))
complex(kind=DCPX)
real*8
```

- wrap assumed-shape arrays

`fwrap` = `f2py` **PLUS...**

Updated features (NB: not all implemented yet!)

- Greater portability – wrap once & distribute, compile on client.
- Extension module is Python 2.x & Python 3.x compatible.
- Automatic resolution of kind-type-parameters:

```
integer(selected_int_kind(0))
complex(kind=DCPX)
real*8
```

- wrap assumed-shape arrays

```
double precision :: arg_array(:,:,:)
```

`fwrap` = `f2py` **PLUS...**

Updated features (NB: not all implemented yet!)

- Greater portability – wrap once & distribute, compile on client.
- Extension module is Python 2.x & Python 3.x compatible.
- Automatic resolution of kind-type-parameters:

```
integer(selected_int_kind(0))
complex(kind=DCPX)
real*8
```

- wrap assumed-shape arrays

```
double precision :: arg_array(:,:,:)
```

- wrap derived types

`fwrap` = `f2py` **PLUS...**

Updated features (NB: not all implemented yet!)

- Greater portability – wrap once & distribute, compile on client.
- Extension module is Python 2.x & Python 3.x compatible.
- Automatic resolution of kind-type-parameters:

```
integer(selected_int_kind(0))
complex(kind=DCPX)
real*8
```

- wrap assumed-shape arrays

```
double precision :: arg_array(:,:,:)
```

- wrap derived types

```
type Point
  real :: x,y
end type Point
```

Motivation for `fwrap`

Motivation for `fwrap`

- I write plasma turbulence simulations as part of my research – FORTRAN is king.

Motivation for `fwrap`

- I write plasma turbulence simulations as part of my research – FORTRAN is king.
- `f2py` has been a great tool, but its age is showing.

Motivation for `fwrap`

- I write plasma turbulence simulations as part of my research – FORTRAN is king.
- `f2py` has been a great tool, but its age is showing.
- It is ideal for F77 code, but has some limitations wrt F9x code (no derived types, no assumed-shape arrays, difficulty handling kind-type-parameters)

Motivation for `fwrap`

- I write plasma turbulence simulations as part of my research – FORTRAN is king.
- `f2py` has been a great tool, but its age is showing.
- It is ideal for F77 code, but has some limitations wrt F9x code (no derived types, no assumed-shape arrays, difficulty handling kind-type-parameters)
- No fault of `f2py`, however: it's impossible to wrap many of these features portably before the 2003 standard, since assumed-shape arrays and derived types are opaque to C.

Motivation for `fwrap`

- I write plasma turbulence simulations as part of my research – FORTRAN is king.
- `f2py` has been a great tool, but its age is showing.
- It is ideal for F77 code, but has some limitations wrt F9x code (no derived types, no assumed-shape arrays, difficulty handling kind-type-parameters)
- No fault of `f2py`, however: it's impossible to wrap many of these features portably before the 2003 standard, since assumed-shape arrays and derived types are opaque to C.
- I always thought it would be great to support these things.

ISO C BINDING and GSoC

ISO C BINDING and GSoC

- The Fortran 2003 standard introduced the `ISO_C_BINDING` intrinsic module that makes wrapping all these extra things pretty easy, along with many other benefits.

ISO C BINDING and GSoC

- The Fortran 2003 standard introduced the `ISO_C_BINDING` intrinsic module that makes wrapping all these extra things pretty easy, along with many other benefits.
- Every major Fortran 9x compiler in existence now supports the `ISO_C_BINDING` module.

ISO C BINDING and GSoC

- The Fortran 2003 standard introduced the `ISO_C_BINDING` intrinsic module that makes wrapping all these extra things pretty easy, along with many other benefits.
- Every major Fortran 9x compiler in existence now supports the `ISO_C_BINDING` module.
- It was Dag Sverre who suggested putting this all together using Cython as the 'glue.'

ISO C BINDING and GSoC

- The Fortran 2003 standard introduced the `ISO_C_BINDING` intrinsic module that makes wrapping all these extra things pretty easy, along with many other benefits.
- Every major Fortran 9x compiler in existence now supports the `ISO_C_BINDING` module.
- It was Dag Sverre who suggested putting this all together using Cython as the 'glue.'
- Add in a little Google Summer of Code support, and you have `fwrap`.

Three levels of wrapping

Three levels of wrapping

- Fortran wrapper

Three levels of wrapping

- Fortran wrapper
  - Can be called from C.

Three levels of wrapping

- Fortran wrapper
  - Can be called from C.
  - Portably!

Three levels of wrapping

- Fortran wrapper
  - Can be called from C.
  - Portably!
  - Use fwrap once, distribute wrappers & source together.

Three levels of wrapping

- Fortran wrapper
  - Can be called from C.
  - Portably!
  - Use fwrap once, distribute wrappers & source together.
  - Can be useful in & of itself.

Three levels of wrapping

- Fortran wrapper
  - Can be called from C.
  - Portably!
  - Use fwrap once, distribute wrappers & source together.
  - Can be useful in & of itself.
- Cython-level wrapper

Three levels of wrapping

- Fortran wrapper
  - Can be called from C.
  - Portably!
  - Use fwrap once, distribute wrappers & source together.
  - Can be useful in & of itself.
- Cython-level wrapper
  - Call the Fortran procedures from Cython.

Three levels of wrapping

- Fortran wrapper
  - Can be called from C.
  - Portably!
  - Use fwrap once, distribute wrappers & source together.
  - Can be useful in & of itself.
- Cython-level wrapper
  - Call the Fortran procedures from Cython.
  - Keeps Python overhead to a minimum.

Three levels of wrapping

- Fortran wrapper
  - Can be called from C.
  - Portably!
  - Use fwrap once, distribute wrappers & source together.
  - Can be useful in & of itself.
- Cython-level wrapper
  - Call the Fortran procedures from Cython.
  - Keeps Python overhead to a minimum.
  - Cython handles the module-building.

Three levels of wrapping (con't)

Three levels of wrapping (con't)

- Python-level that we know and love, like `f2py`.

Three levels of wrapping (con't)

- Python-level that we know and love, like `f2py`.
  - Easy to use

Three levels of wrapping (con't)

- Python-level that we know and love, like `f2py`.
  - Easy to use
  - But small additional overhead.

Three levels of wrapping (con't)

- Python-level that we know and love, like `f2py`.
  - Easy to use
  - But small additional overhead.
  - Especially for fine-grained wrapping.

Three levels of wrapping (con't)

- Python-level that we know and love, like `f2py`.
  - Easy to use
  - But small additional overhead.
  - Especially for fine-grained wrapping.
  - Essentially thin wrappers around the Cython-wrappers.

Three levels of wrapping (con't)

- Python-level that we know and love, like `f2py`.
  - Easy to use
  - But small additional overhead.
  - Especially for fine-grained wrapping.
  - Essentially thin wrappers around the Cython-wrappers.
- All this flexibility adds complexity

Three levels of wrapping (con't)

- Python-level that we know and love, like `f2py`.
  - Easy to use
  - But small additional overhead.
  - Especially for fine-grained wrapping.
  - Essentially thin wrappers around the Cython-wrappers.
- All this flexibility adds complexity
- If you just want a Python extension module that works, it's still (pretty) easy.

Three levels of wrapping (con't)

- Python-level that we know and love, like `f2py`.
  - Easy to use
  - But small additional overhead.
  - Especially for fine-grained wrapping.
  - Essentially thin wrappers around the Cython-wrappers.
- All this flexibility adds complexity
- If you just want a Python extension module that works, it's still (pretty) easy.
- (Once we have distutils/scons automation ironed out, that is...)

Simple Example #1

Simple Example #1

```
function simple_array(int_arr)
    implicit none
    integer(selected_int_kind(10)) :: int_arr(:,:)
    integer(selected_int_kind(10)) :: simple_array

    simple_array = sum(int_arr)

end function simple_array
```

Simple Example #1

```
function simple_array(int_arr)
    implicit none
    integer(selected_int_kind(10)) :: int_arr(:,:)
    integer(selected_int_kind(10)) :: simple_array

    simple_array = sum(int_arr)

end function simple_array
```

Invoke `fwrap`:

```
$ fwrap -projname=fwrap_simple_array simple_array.f90
```

Simple Example #1

```
function simple_array(int_arr)
    implicit none
    integer(selected_int_kind(10)) :: int_arr(:,:)
    integer(selected_int_kind(10)) :: simple_array

    simple_array = sum(int_arr)

end function simple_array
```

Invoke `fwrap`:

```
$ fwrap -projname=fwrap_simple_array simple_array.f90
```

```
$ cd fwrap_simple_array && ls
Makefile                    fwrap_simple_array.pyx
fwrap_setup.py              fwrap_simple_array_fortran.f95
fwrap_simple_array.pxd      fwrap_simple_array_fortran.pxd
simple_array.f95            fwrap_simple_array_header.h
genconfig.f95               setup.py
```

Fortran Wrapper

Fortran Wrapper

```
function fwrap_simple_array(int_arr, int_arr_shape) &
                            bind(c,name="simple_array")
  use config
  implicit none
  integer(kind=fwrap_ardim_long), &
              dimension(2),intent(IN) :: int_arr_shape
  type(c_ptr),value :: int_arr
```

Fortran Wrapper

```
function fwrap_simple_array(int_arr, int_arr_shape) &
                            bind(c,name="simple_array")
  use config
  implicit none
  integer(kind=fwrap_ardim_long), &
              dimension(2),intent(IN) :: int_arr_shape
  type(c_ptr),value :: int_arr

  integer(kind=fwrap_int_slctd_int10), &
                dimension(:,:),pointer :: int_arr_fptr
  integer(kind=fwrap_int_slctd_int10) :: fwrap_simple_array
```

Fortran Wrapper

Fortran Wrapper

```
interface
INTEGER(KIND=selected_int_kind(10)) FUNCTION &
                              simple_array(int_arr)
  implicit none
  integer(selected_int_kind(10)) :: int_arr(:,:)
end function simple_array
end interface
```

Fortran Wrapper

```
  interface
  INTEGER(KIND=selected_int_kind(10)) FUNCTION &
                                simple_array(int_arr)
    implicit none
    integer(selected_int_kind(10)) :: int_arr(:,:)
  end function simple_array
  end interface

  call c_f_pointer(int_arr, int_arr_fptr, int_arr_shape)
  fwrap_simple_array = simple_array(int_arr_fptr)
end function fwrap_simple_array
```

Cython & Python Wrappers

```
cimport fwrap_simple_array_fortran as wf
cimport numpy as np
```

Cython & Python Wrappers

```
cimport fwrap_simple_array_fortran as wf
cimport numpy as np

cdef api wf.fwrap_int_slctd_int10 cy_simple_array(
                        wf.fwrap_int_slctd_int10[:,:] int_arr):
  cdef wf.fwrap_int_slctd_int10[::1,:] work_arr
  if int_arr.is_f_contig():
    work_arr = int_arr
  else:
    work_arr = int_arr.copy_fortran()
```

Cython & Python Wrappers

```
cimport fwrap_simple_array_fortran as wf
cimport numpy as np

cdef api wf.fwrap_int_slctd_int10 cy_simple_array(
                        wf.fwrap_int_slctd_int10[:,:] int_arr):
  cdef wf.fwrap_int_slctd_int10[::1,:] work_arr
  if int_arr.is_f_contig():
    work_arr = int_arr
  else:
    work_arr = int_arr.copy_fortran()

  cdef wf.fwrap_int_slctd_int10 fwrap_return
  cdef wf.fwrap_int_slctd_int10 *int_arr_ptr
  cdef wf.fwrap_ardim_long *int_arr_shape
```

Cython & Python Wrappers

Cython & Python Wrappers

```
int_arr_ptr = work_arr.data
int_arr_shape = <wf.fwrap_ardim_long*>work_arr.shape
```

Cython & Python Wrappers

```
int_arr_ptr = work_arr.data
int_arr_shape = <wf.fwrap_ardim_long*>work_arr.shape

fwrap_return = wf.simple_array(int_arr_ptr, int_arr_shape)
if work_arr is not int_arr:
  int_arr[...] = work_arr
return fwrap_return
```

Cython & Python Wrappers

```
  int_arr_ptr = work_arr.data
  int_arr_shape = <wf.fwrap_ardim_long*>work_arr.shape

  fwrap_return = wf.simple_array(int_arr_ptr, int_arr_shape)
  if work_arr is not int_arr:
    int_arr[...] = work_arr
  return fwrap_return

def simple_array(int_arr):
  cdef wf.fwrap_int_slctd_int10 fwrap_return
  fwrap_return = cy_simple_array(int_arr)
  return (fwrap_return,)
```

Simple Example #2

Simple Example #2

```
function menagerie(int_arg, real_arg, logical_arg, character_arg)
    implicit none
    integer(selected_int_kind(10)), intent(inout) :: int_arg
    real(kind=kind(0.0)), intent(in) :: real_arg
    logical(4) :: logical_arg
    character(len=1) :: character_arg
    integer*8 menagerie

    if(logical_arg) then
        print *, character_arg
    endif
    menagerie = int_arg * real_arg

end function menagerie
```

Simple Example #2

```
function menagerie(int_arg, real_arg, logical_arg, character_arg)
    implicit none
    integer(selected_int_kind(10)), intent(inout) :: int_arg
    real(kind=kind(0.0)), intent(in) :: real_arg
    logical(4) :: logical_arg
    character(len=1) :: character_arg
    integer*8 menagerie

    if(logical_arg) then
        print *, character_arg
    endif
    menagerie = int_arg * real_arg

end function menagerie
```

Invoke `fwrap`:

```
$ fwrap -projname=fw_menagerie menagerie.f95
```

Simple Example #2 Con't.

Simple Example #2 Con't.

- `fwrap` generates files that have no future `fwrap` dependence.

Simple Example #2 Con't.

- `fwrap` generates files that have no future `fwrap` dependence.
  - You ship them and client runs build scripts.

Simple Example #2 Con't.

- `fwrap` generates files that have no future `fwrap` dependence.
  - You ship them and client runs build scripts.

config.f95:

Simple Example #2 Con't.

- `fwrap` generates files that have no future `fwrap` dependence.
  - You ship them and client runs build scripts.

config.f95:

```
module config
  use iso_c_binding
  implicit none
  integer, parameter :: fwrap_int_slctd_int10 = c_long_long
  integer, parameter :: fwrap_real_kind00 = c_float
  integer, parameter :: fwrap_lgcl4 = c_int
  integer, parameter :: fwrap_char_x_1 = c_char
  integer, parameter :: fwrap_int_x_8 = c_long_long
end module config
```

Automatic type resolution

config.h:

```
typedef long long int fwrap_int_slctd_int10;
typedef float fwrap_real_kind00;
typedef int fwrap_lgcl4;
typedef char fwrap_char_x_1;
typedef long long int fwrap_int_x_8;
```

Simple Example Con't.

Simple Example Con't.

- `fw_menagerie_fortran.h` – Along with `config.h` comprise the full C headers for the wrapped Fortran code.

Simple Example Con't.

- `fw_menagerie_fortran.h` – Along with `config.h` comprise the full C headers for the wrapped Fortran code.
- `fw_menagerie_fortran.pxd` – Cython `.pxd` file that declares the C header contents.

Simple Example Con't.

- `fw_menagerie_fortran.h` – Along with `config.h` comprise the full C headers for the wrapped Fortran code.
- `fw_menagerie_fortran.pxd` – Cython `.pxd` file that declares the C header contents.
- `fw_menagerie.pyx` & `fw_menagerie.pxd` – Cython/Python wrapper and associated definition file.

Simple Example Con't.

- `fw_menagerie_fortran.h` – Along with `config.h` comprise the full C headers for the wrapped Fortran code.
- `fw_menagerie_fortran.pxd` – Cython `.pxd` file that declares the C header contents.
- `fw_menagerie.pyx` & `fw_menagerie.pxd` – Cython/Python wrapper and associated definition file.
- `fw_menagerie.pyx` is compiled to C source that becomes the Python extension module.

First release, etc.

First release, etc.

- `fwrap` uses the Fortran parser `fparser`, a subpackage of the `G3 F2PY` project.

First release, etc.

- `fwrap` uses the Fortran parser `fparser`, a subpackage of the `G3 F2PY` project.
  - Generously contributed by Pearu Peterson, author of `f2py`.

First release, etc.

- `fwrap` uses the Fortran parser `fparser`, a subpackage of the `G3 F2PY` project.
  - Generously contributed by Pearu Peterson, author of `f2py`.
  - `fparser` is stabilizing, along with `fwrap`

First release, etc.

- `fwrap` uses the Fortran parser `fparser`, a subpackage of the `G3 F2PY` project.
  - Generously contributed by Pearu Peterson, author of `f2py`.
  - `fparser` is stabilizing, along with `fwrap`
  - Still a ways to go to ensure robustness.

First release, etc.

- `fwrap` uses the Fortran parser `fparser`, a subpackage of the `G3 F2PY` project.
  - Generously contributed by Pearu Peterson, author of `f2py`.
  - `fparser` is stabilizing, along with `fwrap`
  - Still a ways to go to ensure robustness.
- First production release will focus on wrapping large F77 packages (of the level of BLAS/LAPACK) and basic features of F9x programs.

First release, etc.

- `fwrap` uses the Fortran parser `fparser`, a subpackage of the `G3 F2PY` project.
  - Generously contributed by Pearu Peterson, author of `f2py`.
  - `fparser` is stabilizing, along with `fwrap`
  - Still a ways to go to ensure robustness.
- First production release will focus on wrapping large F77 packages (of the level of BLAS/LAPACK) and basic features of F9x programs.
  - Fully support all intrinsic types, and their arrays.

First release, etc.

- `fwrap` uses the Fortran parser `fparser`, a subpackage of the `G3 F2PY` project.
  - Generously contributed by Pearu Peterson, author of `f2py`.
  - `fparser` is stabilizing, along with `fwrap`
  - Still a ways to go to ensure robustness.
- First production release will focus on wrapping large F77 packages (of the level of BLAS/LAPACK) and basic features of F9x programs.
  - Fully support all intrinsic types, and their arrays.
  - Assumed-shape, assumed-size and explictly-shaped arrays.

First release, etc.

- `fwrap` uses the Fortran parser `fparser`, a subpackage of the `G3 F2PY` project.
  - Generously contributed by Pearu Peterson, author of `f2py`.
  - `fparser` is stabilizing, along with `fwrap`
  - Still a ways to go to ensure robustness.
- First production release will focus on wrapping large F77 packages (of the level of BLAS/LAPACK) and basic features of F9x programs.
  - Fully support all intrinsic types, and their arrays.
  - Assumed-shape, assumed-size and explictly-shaped arrays.
  - Fully automatic resolution of kind-type-parameters.

First release, etc.

- `fwrap` uses the Fortran parser `fparser`, a subpackage of the `G3 F2PY` project.
  - Generously contributed by Pearu Peterson, author of `f2py`.
  - `fparser` is stabilizing, along with `fwrap`
  - Still a ways to go to ensure robustness.
- First production release will focus on wrapping large F77 packages (of the level of BLAS/LAPACK) and basic features of F9x programs.
  - Fully support all intrinsic types, and their arrays.
  - Assumed-shape, assumed-size and explictly-shaped arrays.
  - Fully automatic resolution of kind-type-parameters.
  - ETA: before 2010, November/December 2009.

Wishlist & BoF

Wishlist & BoF

- C, Cython & Python callbacks in Fortran code

Wishlist & BoF

- C, Cython & Python callbacks in Fortran code
  - Not in first distributed version.

Wishlist & BoF

- C, Cython & Python callbacks in Fortran code
  - Not in first distributed version.
- Re-entrancy & threading issues

Wishlist & BoF

- C, Cython & Python callbacks in Fortran code
  - Not in first distributed version.
- Re-entrancy & threading issues
- Better `ISO C BINDING` support in `gfortran`

Wishlist & BoF

- C, Cython & Python callbacks in Fortran code
  - Not in first distributed version.
- Re-entrancy & threading issues
- Better `ISO C BINDING` support in `gfortran`
- What do you want to see in fwrap?

Wishlist & BoF

- C, Cython & Python callbacks in Fortran code
  - Not in first distributed version.
- Re-entrancy & threading issues
- Better `ISO C BINDING` support in `gfortran`
- What do you want to see in fwrap?
- Let me know – lunch, break, or Cython/Fwrap BoF.